The Beer Dock:
Three and a Half Implementations of the Beer Distribution Game

Michael J. North[1] and Charles M. Macal
Argonne National Laboratory, Argonne, Illinois

Abstract

The Beer Distribution Game is a classic supply chain problem widely used in graduate business programs to teach the concepts of supply chain management (Moseklide, Larsen, and Sterman, 1991). It is well suited for this purpose since it is simple enough to be easily understandable but complex enough to be interesting. In particular, the Beer Distribution Game is notable for its ability to confound typical human players (Sterman, 1987, 1989). Many people who play the game find it difficult, if not impossible, to avoid the chaotic operating regimes that are the game's hallmark.

According to Burton, docking "is a compelling metaphor from space exploration" that "offers much promise to give simulation modeling greater validity" (1998). Docking is used to "determine whether two models can produce the same results, which in turn is the basis for critical experiments and for tests of whether one model can subsume another" (Axtell, Axelrod, Epstein, and Cohen, 1996).

The original Beer Distribution Game was implemented as a systems dynamics model. The authors have implemented the game using Mathematica functional programming, RePast agent-based modeling and simulation (ABMS), and Swarm ABMS. The authors have reproduced all the published results with these new implementations. As part of this process, the authors have docked the implementations. The docking process was found to be challenging and time consuming, but ultimately rewarding.

The Beer Distribution Game

The Beer Distribution Game ("Beer Game") is a classic supply chain problem widely used in graduate business programs to teach the concepts of supply chain management (Moseklide, Larsen, and Sterman, 1991). It is well suited for this purpose since it is simple enough to be easily understandable but complex enough to be interesting. In particular, the Beer Game is notable for its ability to confound typical human players (Sterman, 1987, 1989). Many people who play the game find it difficult, if not impossible, to avoid the chaotic operating regimes that are the game's hallmark.

The Beer Game consists of a supply chain and five agents as shown in Figure 1. The customer places an order with the retailer who fills the order if the retailer's inventory allows. The retailer orders additional items from the wholesaler. There is a one-period delay in the order being received. The wholesaler fills the order if the wholesaler's inventory allows. There is a two-period delay in items being shipped and reaching their

[1] *Corresponding author address*: Michael J. North, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439; e-mail: north@anl.gov; Telephone: (630) 252-6234; Facsimile: (630) 252-6073

destination. The wholesaler orders additional items from the distributor and so on for the distributor and factory. If the factory cannot fill an order it places the order in production. There is a three-period production delay. Initially, the supply chain is in complete equilibrium in terms of demand, orders, supplies, and inventory.

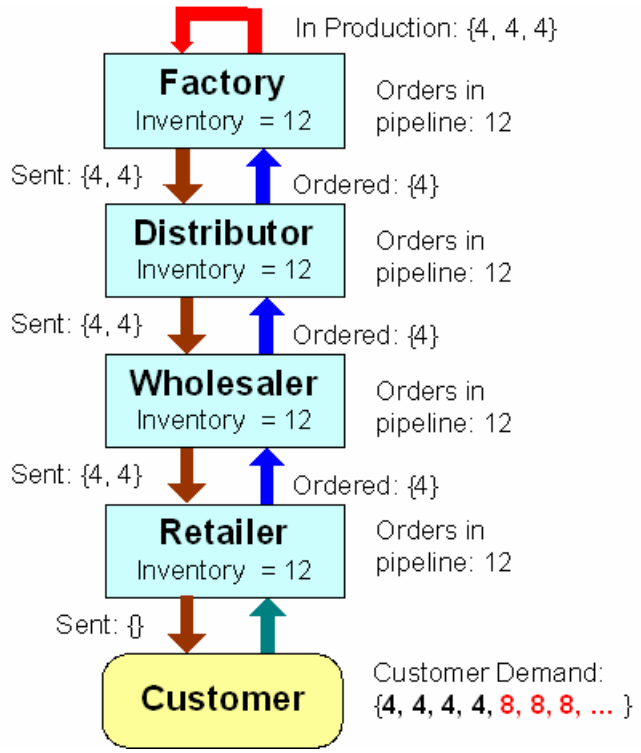


Figure 1: The Beer Distribution System

In the Beer Game, each agent makes ordering decisions based only on locally available information. Orders are based on the following factors:

- Inventory
- Desired inventory level
- Items in the pipeline (shipments in-transit from upstream agent and outstanding orders placed to agent upstream)
- Desired level of items in the pipeline
- Current demand
- Expected demand

Agents have the goal of achieving desired inventory and pipeline levels. Agents seek to close the gap between desired and actual levels in terms of their stock and what they have in the pipeline. Stock adjustments to inventory are determined as:

$$\text{Stock Adjustment to Inventory}_t = a_S\,(\text{Desired Inventory} - \text{Inventory}_t),$$

Stock adjustments to the pipeline are determined as:

$$\text{Stock Adjustment to Pipeline}_t = a_{SL} (\text{Desired Pipeline} - \text{Pipeline}_t).$$

Ordering parameters a_S and a_{SL} represent the fraction of the gap to be closed between desired and actual levels for each order. The ordering parameter $b = (a_{SL} / a_S)$ is the relative weight attached to pipeline versus stock discrepancies from desired levels. Agents adjust their demand projections each period. Agents weight the current demand and the expected demand for this period to estimate the demand for the next period:

$$\text{Expected Demand}_{t+1} = q\ \text{Demand}_t + (1 - q)\ \text{Expected Demandt}$$

where $0 <= q <= 1$. Agents use an adaptive behavioral decision rule to determine orders:

$$\text{Indicated Order}_t = \text{Expected Demand}_t + \text{Stock Adjustment Inventory}_t + \text{Stock Adjustment Pipeline}_t$$

That is,

$$\text{Indicated Order}_t = ED_t + a_S (Q - \text{Inventory}_t - b * \text{Items in Pipeline}_t)$$

where Q is a measure of desired inventory relative to desired pipeline:

$$Q = \text{Desired Inventory} + b * \text{Desired Pipeline Line}$$

Finally since negative orders are not allowed, the order the agent places to its upstream agent at t is:

$$\text{Order}_t = \text{Max}(0, \text{Indicated Order}_t)$$

Arguably, this is a good, descriptive, behavioral decision model for this context (Sterman 1987, Sterman 1989).

The Beer Game system is completely deterministic. There are no random elements in the model. If demand does not change, the system continues forever in complete equilibrium:

- At every stage orders received equal orders sent
- For all stages the pipeline and inventory values are 12
- The customer demand of four units per period is always satisfied

At time five, a change occurs:

- Demand ramps up to 8 per period and stays there

- A new ordering rule takes effect that determines how much agents at each stage of the supply chain will order, based on the locally available information

The system is then simulated forward from this point onward, ultimately leading to deterministic chaos in many cases.

The Mathematica Experiments

To investigate the Beer Game, an implementation was constructed using Wolfram Research's Mathematica®. Mathematica is an integrated, highly programmable, technical computing environment (Wolfram 2002).

The goal was to duplicate the results of described in Mosekilde, Larsen, and Sterman (MLS)(1991). MLS found that varying the ordering parameters of the model results in various modes of long-run system behavior. They identified behaviors as stable, chaotic, periodic (non-chaotic) and quasi-periodic (chaotic) over the range of parameter values.

The MLS results were reproduced exactly for the case of $a_s = 0.3$, $b = 0.15$, $q = 0$, $Q = 15$ as shown in Figures 2 and 3. Figure 3 is comparable to MLS Figure 5.

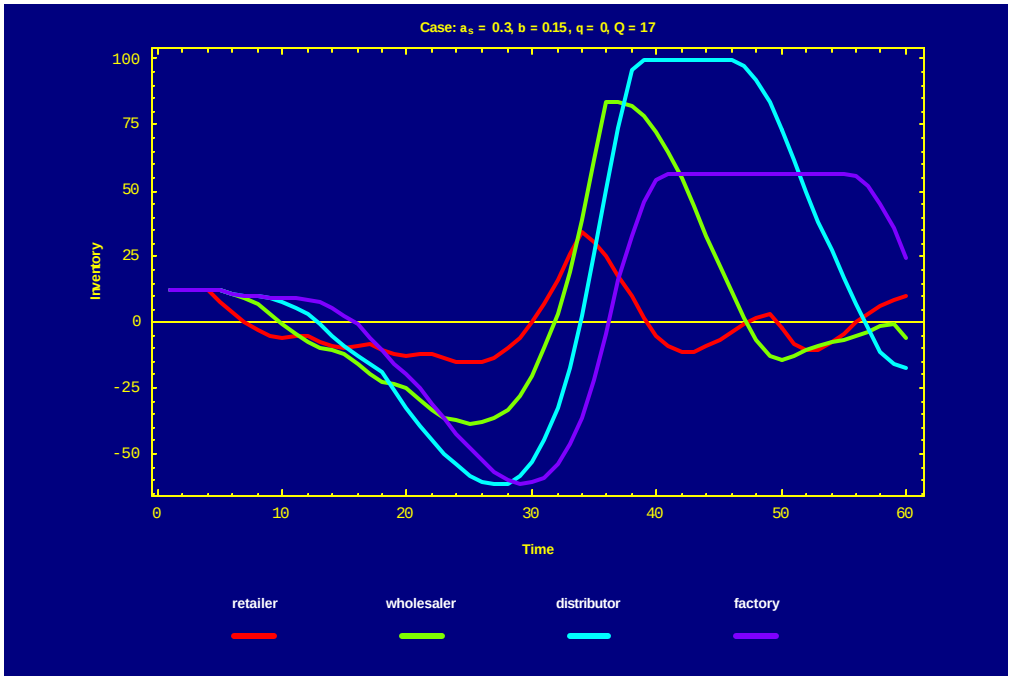


Figure 2: The 60 Period Mathematica Beer Game Inventory Graph for $a_s = 0.3$, $b = 0.15$, $q = 0$, $Q = 15$

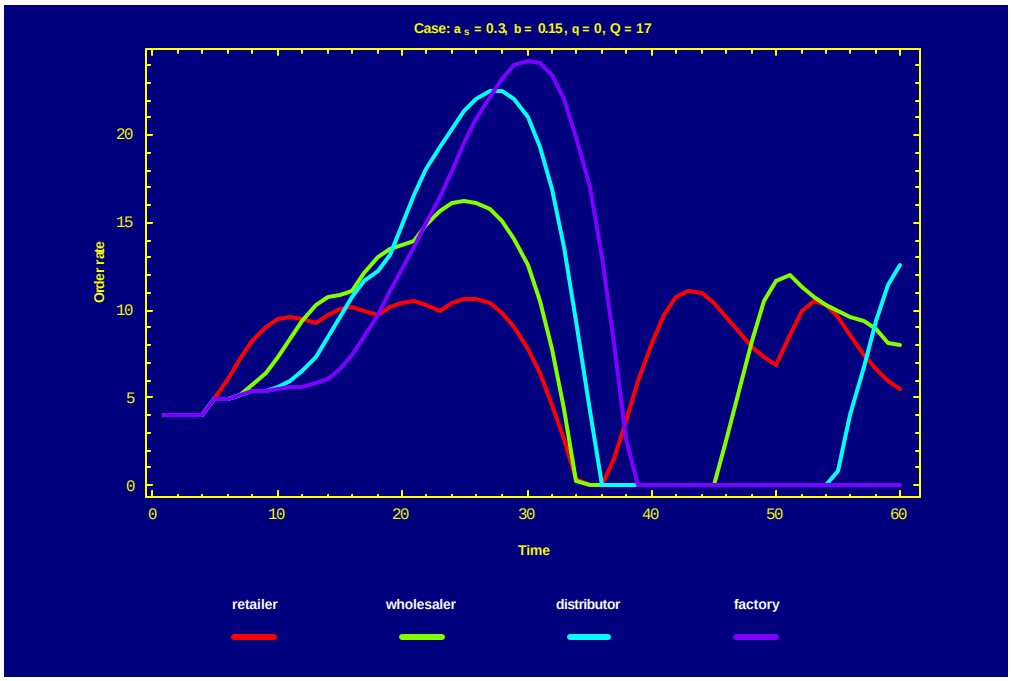


Figure 3: The 60 Period Mathematica Beer Game Order Rate Graph for a_s = 0.3, b = 0.15, q = 0, Q = 15 (Comparable to MLS Figure 5)

Figures 4 and 5 plot the same case over 1,000 time periods. Figure 5 is comparable to MLS Figure 6.

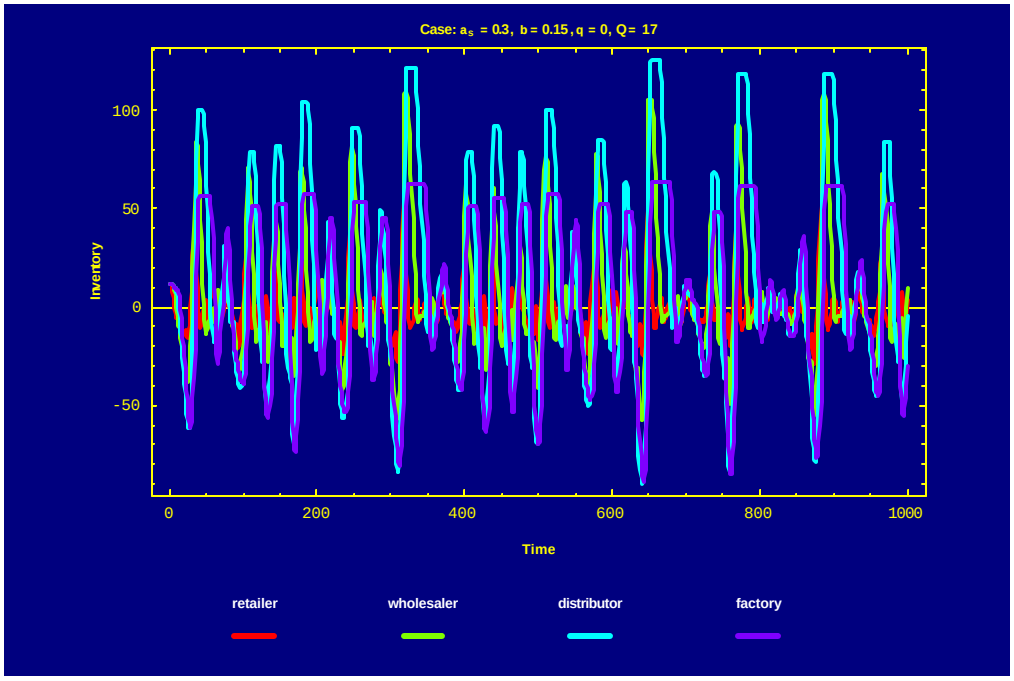


Figure 5: The 1,000 Period Mathematica Beer Game Inventory Graph for a_s = 0.3, b = 0.15, q = 0, Q = 15

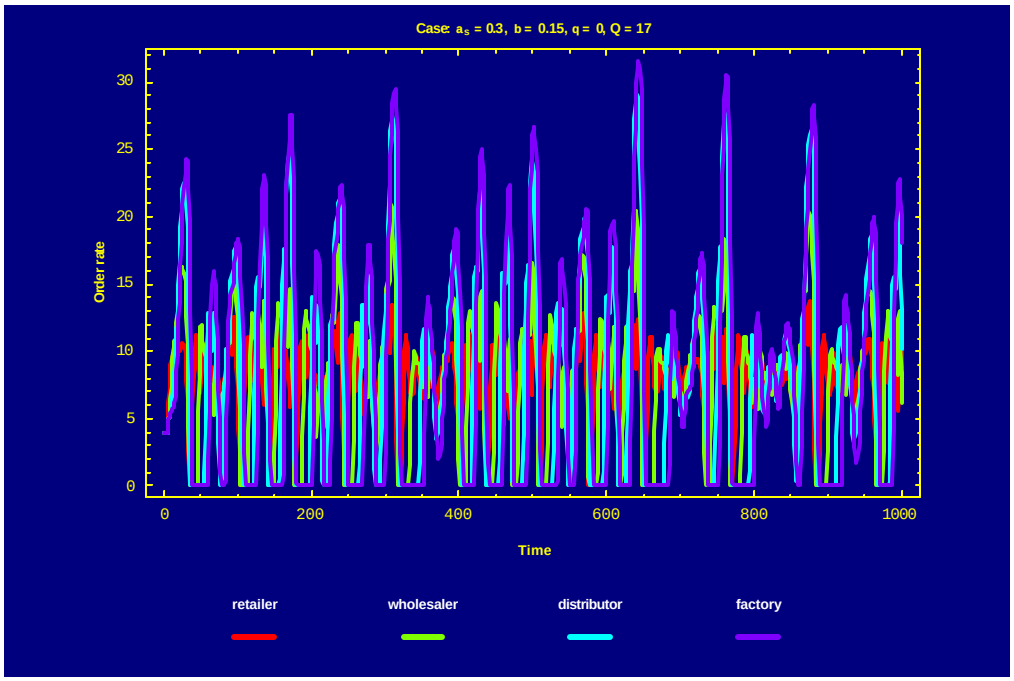


Figure 6: The 1,000 Period Mathematica Beer Game Order Rate Graph for a_s = 0.3, b = 0.15, q = 0, Q = 15 (Comparable to MLS Figure 6)

For the case of a_s = 0.5, b = 0, q = 0.05, Q = 15, very similar but not exact results were obtained as shown in Figures 7 and 8. Figure 8 is comparable to MLS Figure 7.

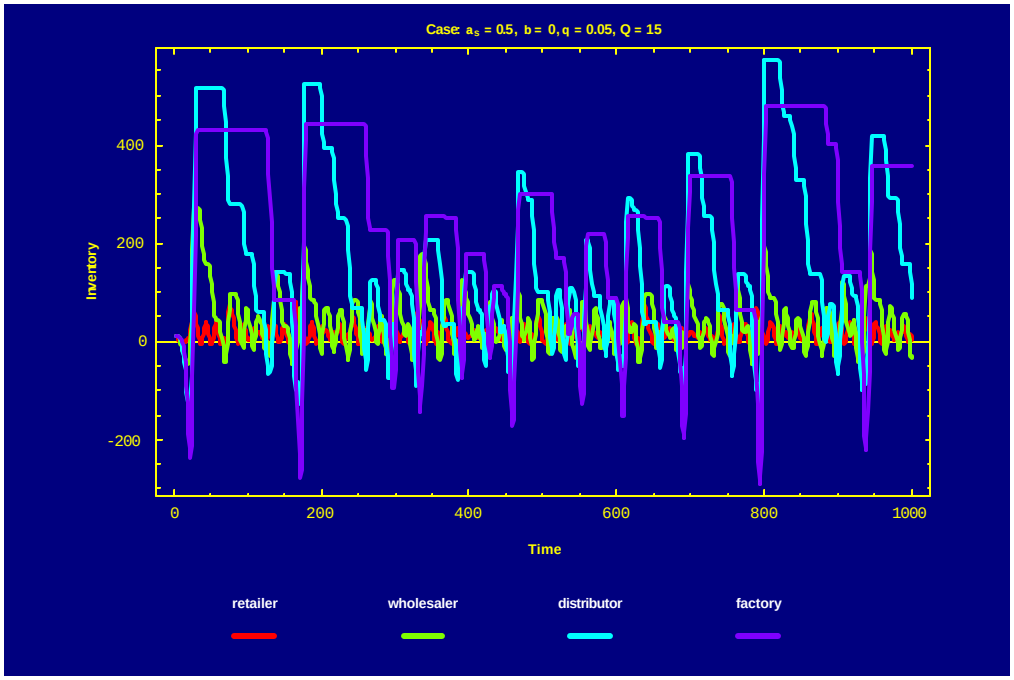


Figure 7: The 1,000 Period Mathematica Beer Game Inventory Graph for a_s = 0.5, b = 0, q = 0.05, Q = 15

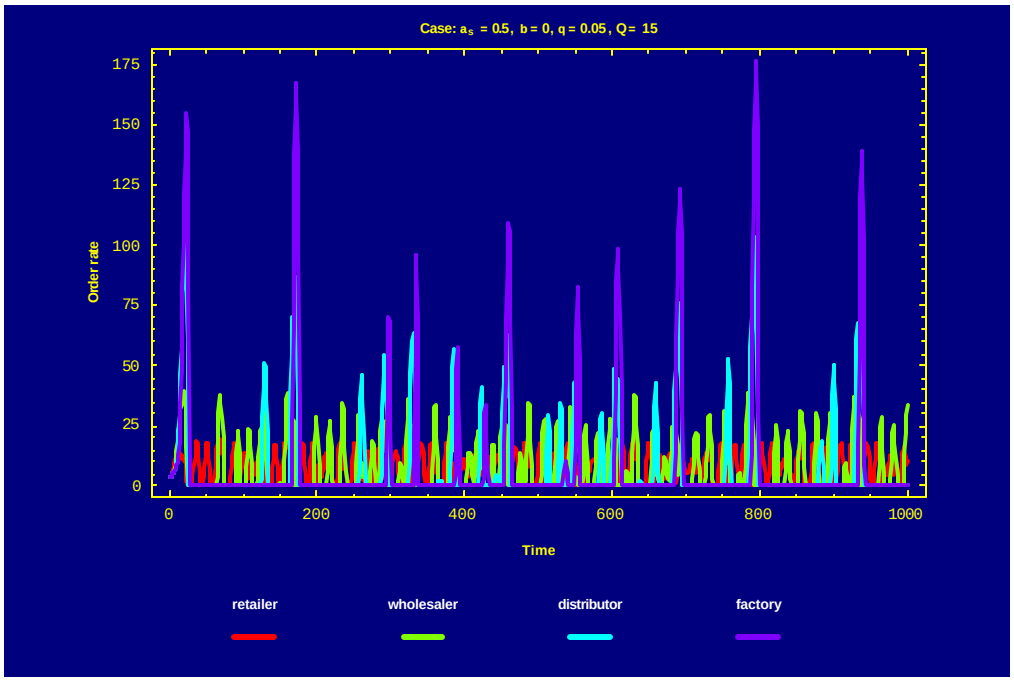


Figure 8: The 1,000 Period Mathematica Beer Game Order Rate Graph for a_s = 0.3, b = 0.15, q = 0, Q = 15 (Comparable to MLS Figure 7)

For the case of a_s = 0.3, b = 0.65, q = 0.25, Q = 12, the same results were obtained as shown in Figure 9. MLS classifies the result as periodic, although technically the series is non-repeating and exhibits chaotic behavior. Figure 9 is comparable to MLS Figure 8. This is described by MLS as a periodic trajectory with a transient that has a regular damped approach to a limit cycle. The Mathematica analysis program was adjusted to classify this series as quasi-periodic.

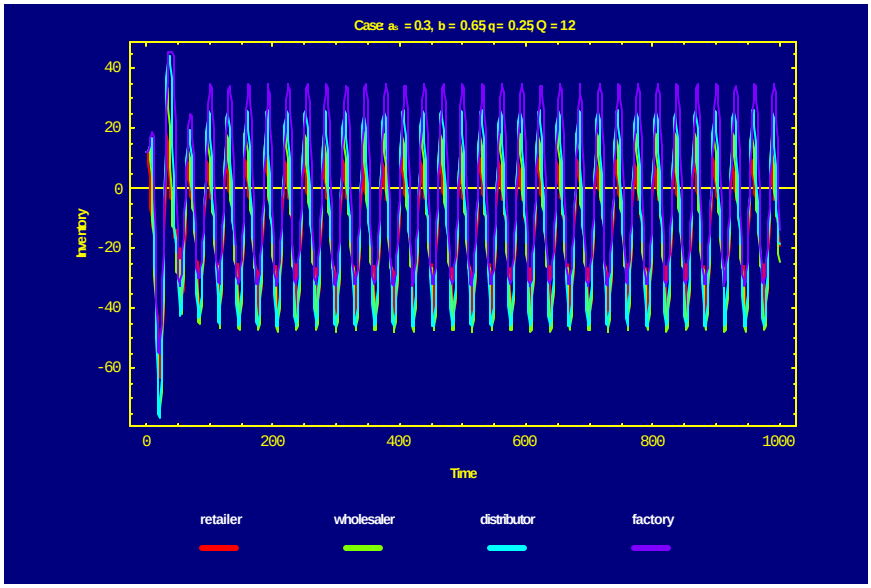


Figure 9: The 1,000 Period Mathematica Beer Game Inventory Graph for a_s = 0.3, b = 0.65, q = 0.25, Q = 12 (Comparable to MLS Figure 8)

For the case of $a_S = 0.425$, b = 0.09, q = 0.25, Q = 17, somewhat different results were obtained as shown in Figure 10. This Figure is comparable to MLS Figure 9. MLS classified his results as a period-4 limit cycle, but the Mathematica results are chaotic with no clear limit cycles observed.

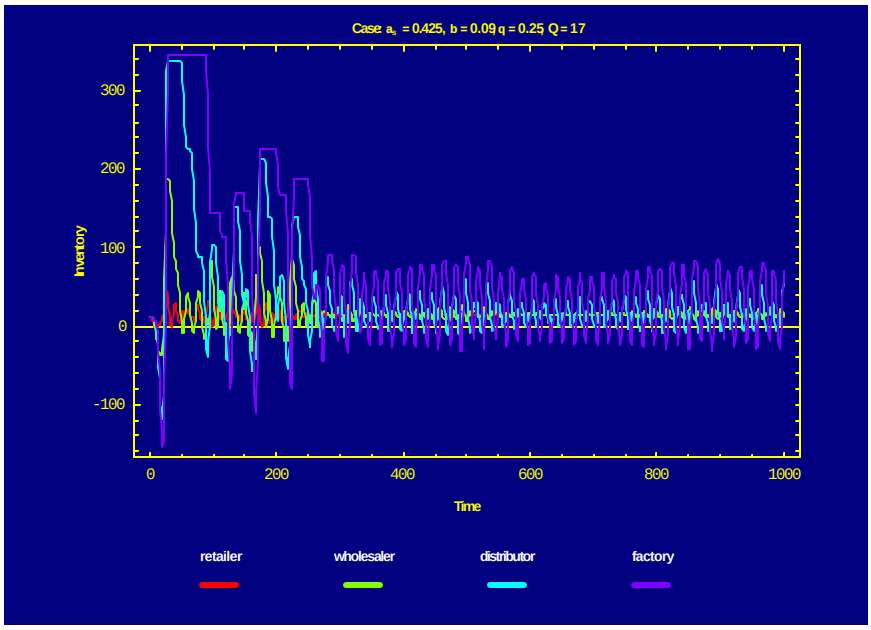


Figure 10: The 1,000 Mathematica Period Beer Game Inventory Graph for $a_S = 0.5$, b = 0, q = 0.05, Q = 15 (Comparable to MLS Figure 9)

The phase plot shown in Figure 11 is similar to that obtained by MLS but does not exhibit their period-4 limit cycle. Rather, Figure 11 exhibits a chaotic attractor.

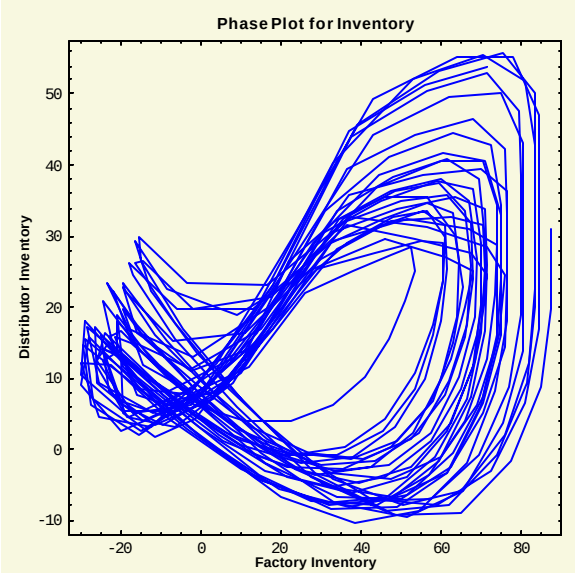
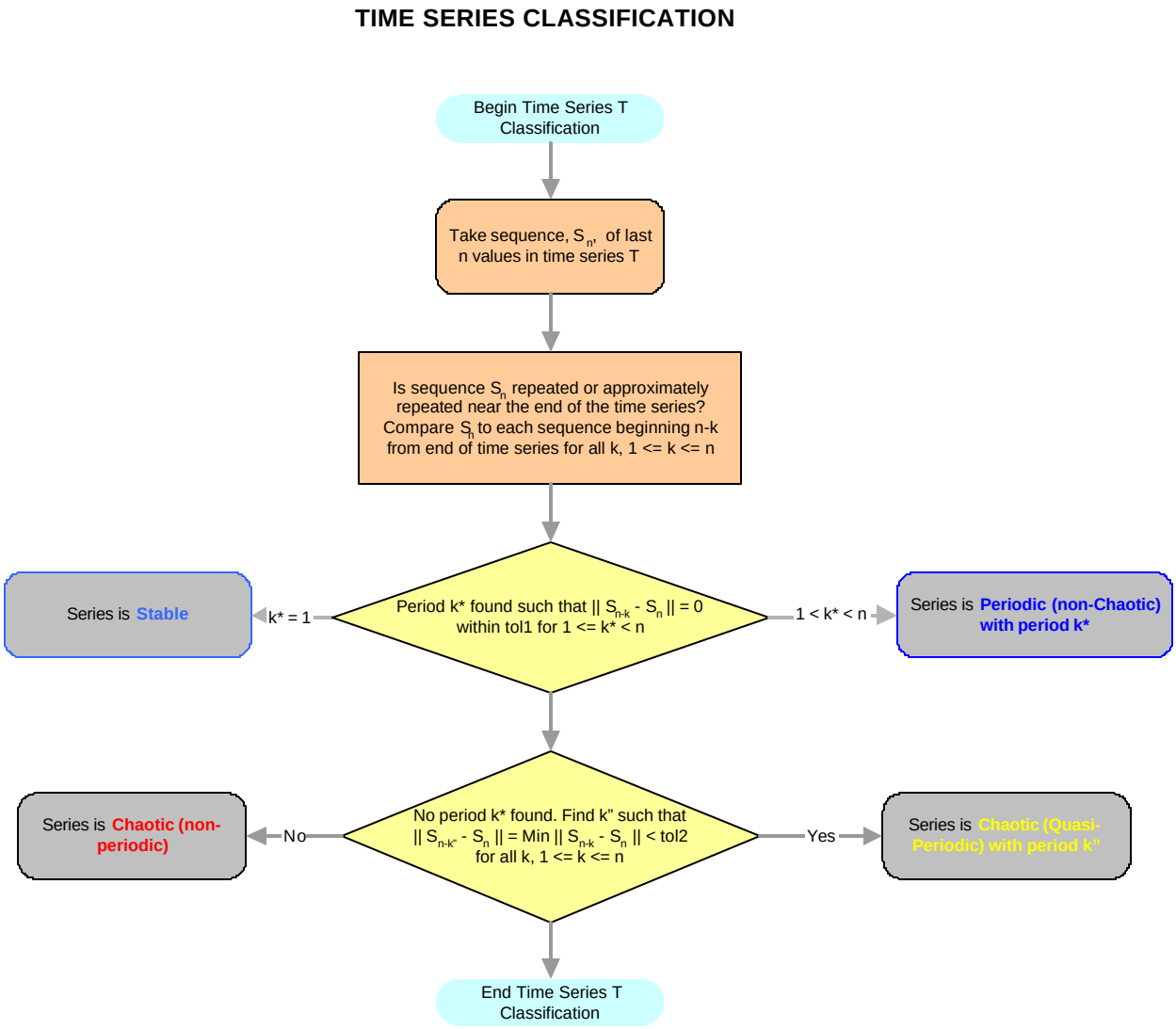


Figure 11: The Mathematica Beer Game Phase Plot for $a_S = 0.425$, b = 0.09, q = 0.25, Q = 17

Four behavior modes have been observed in the long-run time series of the Mathematica Beer Game inventory levels: stable, chaotic, chaotic (quasi-periodic), and periodic. The chaotic modes are non-repeating infinite sequences. In all cases observed, all time series including all chaotic sequences were bounded. Transient behaviors leading up to these long-run classifications were mixed. Figure 12 illustrates the logic used for classifying the time series.



* Notes: (1) n is set at 100.
(2) tol1 is set so values are equivalent if within 2nd decimal place.
(3) tol2 = 2 captures quasi-periodic sequences adequately.

Figure 12: Mode Classification Scheme

A set of 100 x 100 simulations were run over the parameter space (a_s, b) å [0, 1] x [0, 1] for q = 0.25, Q = 17. Each simulation was run for 1,000 periods. The behavior mode for each factory inventory series was identified based on the last 100 values in the series. The distribution of modes shown in Figure compares favorably with MLS Figure 10, allowing for the difference in resolution between the two analyses (MLS conducted 200 x 200 simulations for the same interval). The Figure is read as follows:

- Light blue (medium gray) indicates stable behavior
- Red (dark gray) indicates chaotic behavior
- Dark blue (black) indicates periodic (non-chaotic) behavior
- Yellow (light gray) indicates quasi-periodic, chaotic behavior

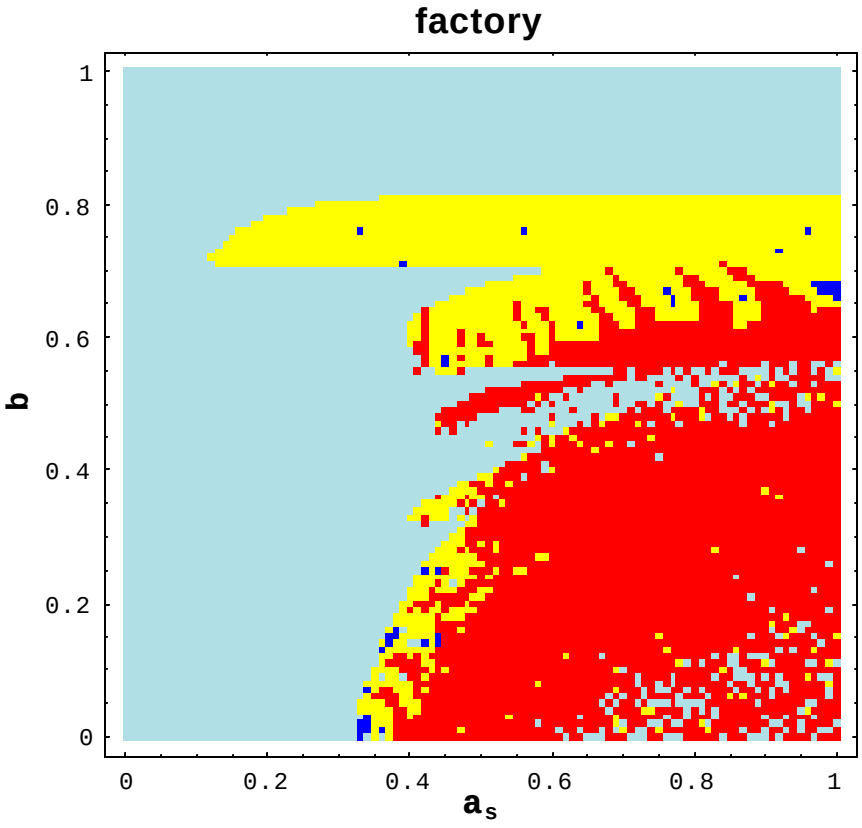


Figure 13: The Mathematica Beer Game Behavior modes in the a_S-b plane (Comparable to MLS Figure 10)

The costs obtained over the policy space are shown in Figure 14. White indicates cost minimums. Lighter shades indicate cost near minimum. Darker shades indicate costs near maximum. As can be seen in the Figure, minimum cost policies are distributed throughout the policy space, often residing in close proximity to high-cost policies.

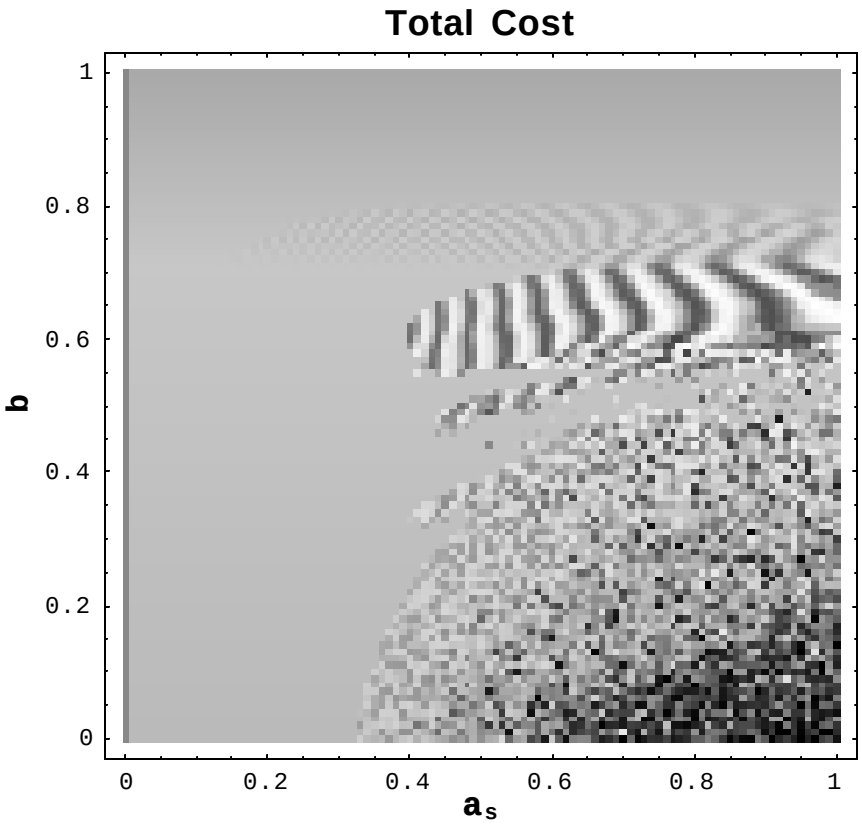


Figure 14: The Mathematica Beer Game Costs in the a_S-â plane (Comparable to MLS Figure 12)

Onward to RePast

The original Beer Game was implemented as a systems dynamics model. As previously discussed, the authors have implemented the game using Mathematica functional programming. The Beer Game was then implemented using the RePast agent-based modeling and simulation (ABMS) toolkit as shown in Figure 15. From the RePast web site (University of Chicago, 2002):

> The University of Chicago's Social Science Research Computing's RePast is a software framework for creating agent based simulations using the Java language (requires version Java 1.3 or greater). It provides a library of classes for creating, running, displaying and collecting data from an agent based simulation. In addition, RePast can take snapshots of running simulations, and create quicktime movies of simulations. RePast borrows much from the Swarm simulation toolkit and can properly be termed "Swarm-like." In addition, RePast includes such features as run-time model manipulation via gui widgets first found in the Ascape simulation toolkit.

The resulting model run for $a_S = 0.3$, b = 0.15, q = 0, Q = 15 is shown in Figures 16 and 17.

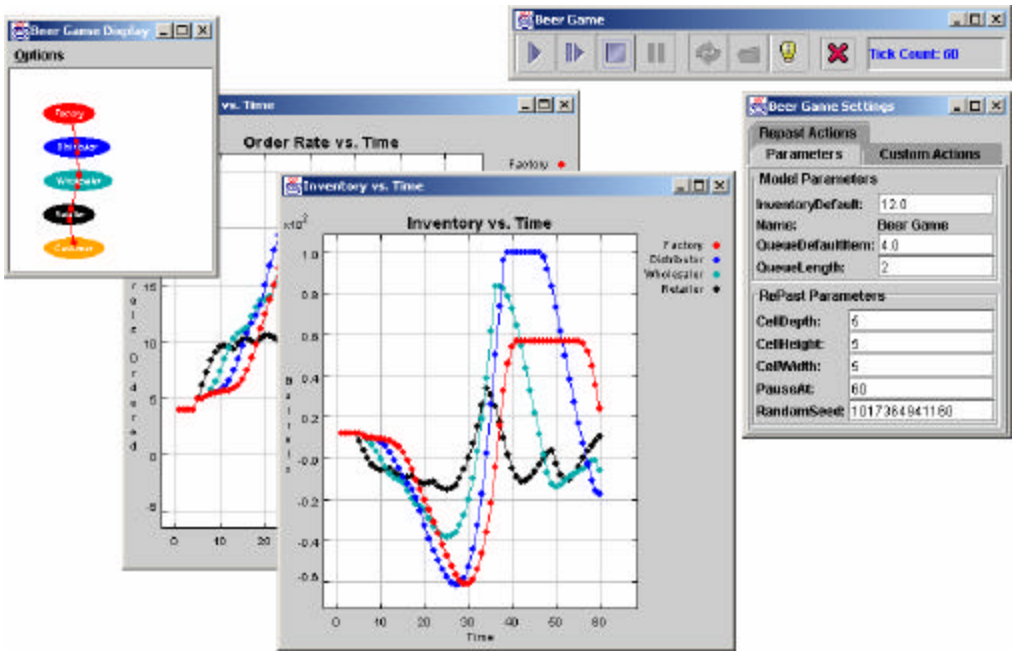


Figure 15: The RePast Beer Game Implementation with $a_s = 0.3$, b = 0.15, q = 0, Q = 15

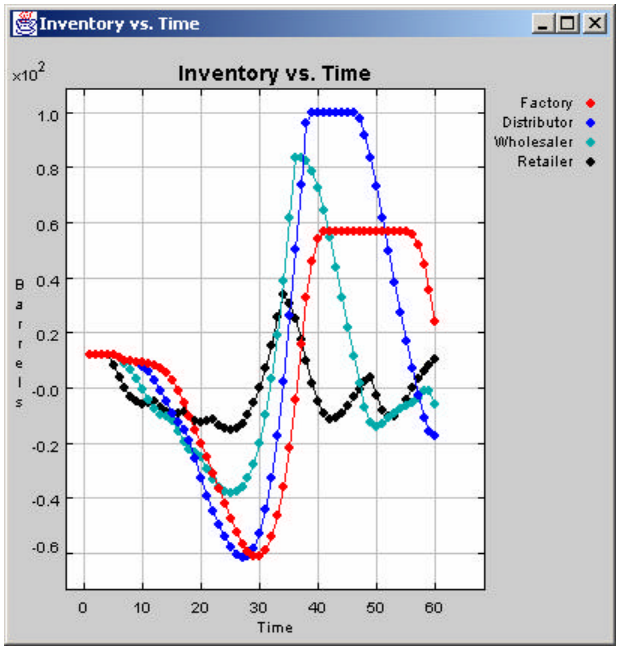


Figure 16: The 60 Period RePast Beer Game Inventory Graph for $a_s = 0.3$, b = 0.15, q = 0, Q = 15

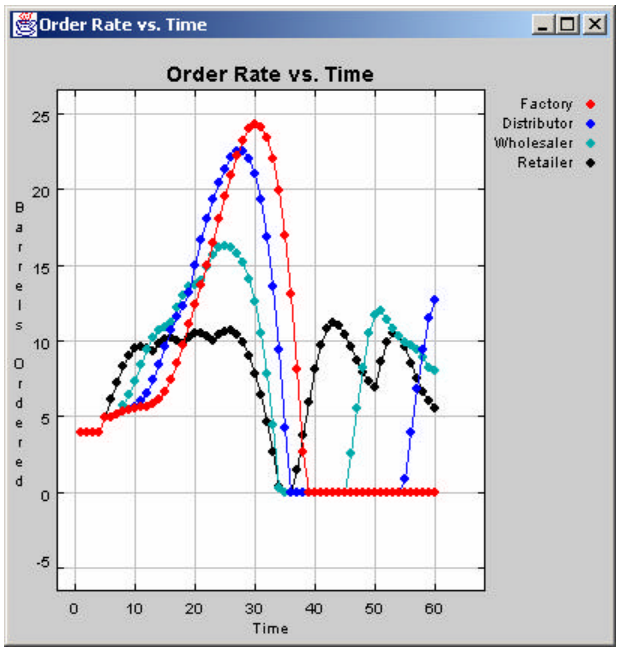


Figure 17: The 60 Period RePast Beer Game Order Rate Graph for a_s = 0.3, b = 0.15, q = 0, Q = 15 (Comparable to MLS Figure 5)

Continuing to Java Swarm

After the RePast implementation, the Beer Game was implemented again using the Java Swarm ABMS toolkit as shown in Figure 18. The resulting model run for a_S = 0.3, b = 0.15, q = 0, Q = 15 is shown in Figures 19 and 20. The Java Swarm version of the Beer Game is counted as "half an implementation" since much of its code is shared with the RePast implementation. According to the Swarm Development Group's web site, "Swarm is a software package for multi-agent simulation of complex systems" (2002).

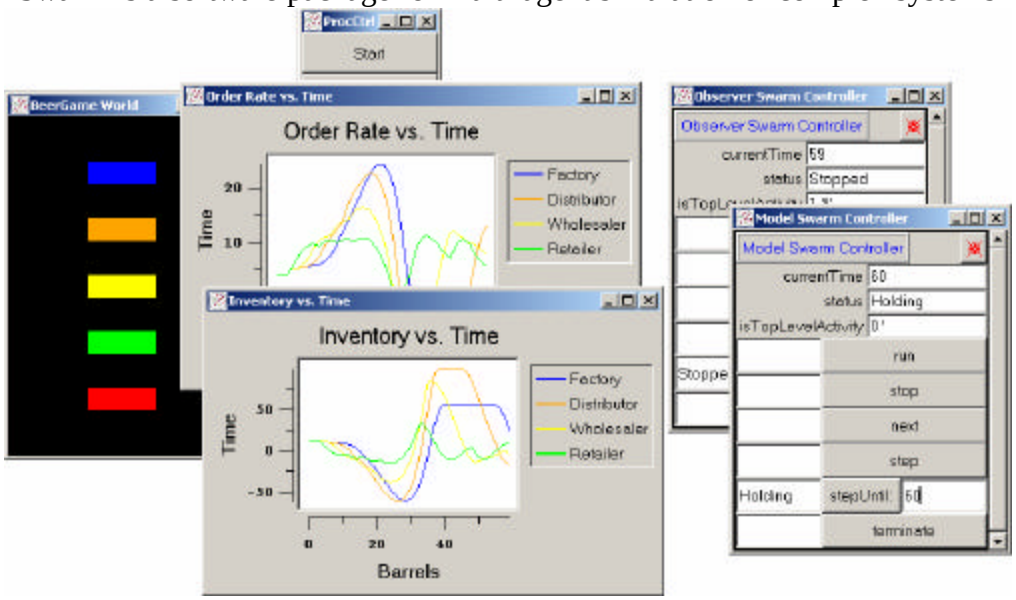


Figure 18: The Swarm Beer Game Implementation with a_s = 0.3, b = 0.15, q = 0, Q = 15

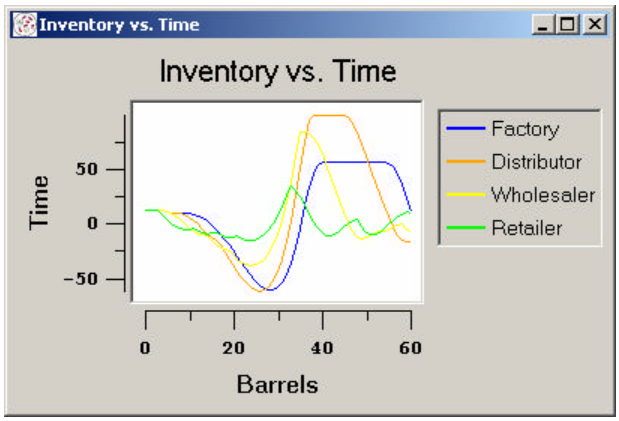


Figure 19: The 60 Period RePast Beer Game Inventory Graph for $a_s = 0.3$, b = 0.15, q = 0, Q = 15

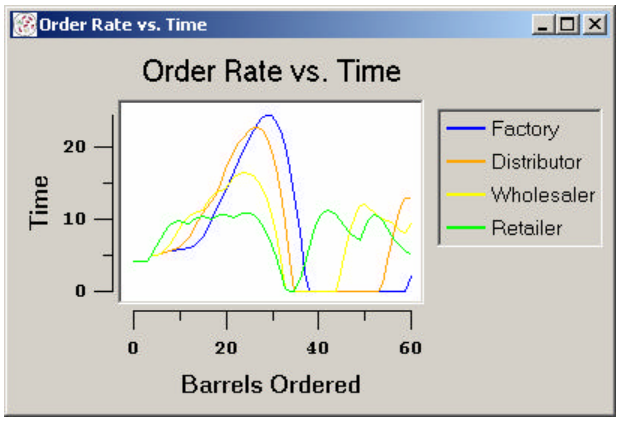


Figure 20: The 60 Period RePast Beer Game Order Rate Graph for $a_s = 0.3$, b = 0.15, q = 0, Q = 15 (Comparable to MLS Figure 5)

Docking

According to Burton, docking "is a compelling metaphor from space exploration" that "offers much promise to give simulation modeling greater validity" (1998). Later Burton wrote the following about docking any two models (1999):

> Docking, or model alignment, is an approach to validation that can give us greater confidence in both models. The ideas is to compare models in a basic way to see how they are similar and different and, more importantly, to increase our confidence that both models can be used to say something about the question under study.

Docking is used to "determine whether two models can produce the same results, which in turn is the basis for critical experiments and for tests of whether one model can subsume another" (Axtell, Axelrod, Epstein, and Cohen, 1996). Docking is thus a verification process that seeks to find isomorphic relations between two or more related models.

Following this approach, the original systems dynamics implementation, the Mathematica implementation, the RePast implementation, and the Java Swarm implementations have where docked. In particular, the key abstractions of the models where compared and contrasted.

The original systems dynamics version is counted as full implementation. Since this version relied on systems dynamics it used direct mathematical assignment operators to produce new values. This seems to have lead to unusual ordering rules that depend only on the previous state and ignore the current knowledge. In particular, using old pipeline and stock numbers to compute orders instead of the more reasonable approach of using the current inputs to calculate the next outputs is a key to matching the model output using other techniques. The key abstractions including the following:

- Scheduling is implemented using time index variables. All of these variables are updated simultaneously, leading to the unusual ordering rules described above.
- Agents are represented as collections of uncoupled variables.

The Mathematica version is a full implementation since it uses a functional style of programming that is significantly different from the original systems dynamics approach. The key abstractions including the following:

- Scheduling is implemented using simple iterated function calls. Variables can be updated sequentially within a time step. The original systems dynamics model's unusual ordering rules must be imposed by buffering values during a time step.
- Agents are represented as collections of loosely coupled variables.

Writing the initial Mathematica model was quite easy. Docking the Mathematica model to the original systems dynamics by mapping corresponding abstractions was extremely time consuming since the only available information on the original model is given in the referenced papers (MLS 1991; Sterman, 1987; Sterman, 1989). Finding an exact match between the models required many repeated runs to uncover and remove the subtle variations between the two models.

The RePast version is a full implementation since it uses an object-oriented style of programming that is significantly different from the previous approaches. The key abstractions including the following:

- Scheduling is implemented using a basic ABMS scheduling system. As with Mathematica, variables can be updated sequentially within a time step. The original systems dynamics model's unusual ordering rules must be imposed by buffering values during a time step.
- Agents are represented as objects.

As with writing the Mathematica model, creating the initial RePast model was easy. Docking the model was much more difficult.

During the RePast docking a substantial amount of time was spent matching the exact numeric outputs from the Mathematica version. After a substantial period of initial work, an exact match was found with the first twenty or so Mathematica model run steps. Later time steps seemed to slightly diverge from the Mathematica values and the systems dynamics charts. At first this seemed to be the result of numerical round off errors. However, increasing the number of Mathematica model output digits from six to 17 revealed a slight difference in the second time step. After a substantial period of inspection this difference was traced to the order of calculation execution. Correcting this order allowed an exact, full precision match to be made and suggests an important observation: Never blame numerical errors unless you can *prove* they are the problem!

As stated previously, the Java Swarm version of the Beer Game is counted as half an implementation since much of its object-oriented Java code is shared with the RePast implementation.

- Scheduling is implemented using a rich ABMS scheduling system. As with Mathematica and RePast, variables can be updated sequentially within a time step. The original systems dynamics model's unusual ordering rules must be imposed by buffering values during a time step.
- Agents are represented as objects.

As with writing the Mathematica model, creating the initial Java Swarm model was easy. The lesions learned from the RePast model work simplified the docking process enough to make it quite manageable.

Conclusion

The docking process was found to be challenging and time consuming, but ultimately rewarding. All of the model implementations where able to produce the same results after a substantial amount of effort was expended. The docking process was found to be particularly valuable since it revealed hidden assumptions embedded in the underlying models. In particular, it was found that ostensibly identical models can have subtle variations that are only revealed upon extremely close inspection.

References

Axtell, R., R. Axelrod, J. Epstein, and M. Cohen. (1996). "Aligning Simulation Models: A Case Study and Results," Computational and Mathematical Organization Theory, 1(2): 123-141.

Richard Burton, (1998) "Validating and Docking: An Overview, Summary and Challenge," M. Prietula, K. Carley & L. Gasser, (editors), Simulating Societies: Computational Models of Institutions and Groups, AAAI/MIT Press, Cambridge, MA.

Richard Burton, (October 1999) “The Challenge of Validation and Docking,” Proceedings of the 1999 Agent Simulation Workshop, University of Chicago, Chicago, IL.

Moseklide, E., E. Larsen, and J. Sterman. (1991). “Coping with Complexity: Deterministic Chaos in Human Decisionmaking Behavior,” In Casti, J. L. and A. Karlqvist. (Eds.), Beyond Belief: Randomness, Prediction, and Explanation in Science, CRC Press, Boston.

Sterman, J. D. (December 1987). “Testing Behavioral Simulation Models by Direct Experiment,” Management Science, 33(12): 1572-1592.

Sterman, J. D. (March 1989). “Modeling Managerial Behavior: Misperceptions of Feedback in a Dynamic Decision Making Experiment,” Management Science, 35(3): 321-339.

Swarm Development Group (March 2002). Swarm Development Group Web Site, Available as http://www.swarm.org/.

University of Chicago (March 2002). RePast Web Site, Available as http://repast.sourceforge.net /.

Wolfram Research (March 2002). Mathematica Web Site, Available as http://www.wolfram.com/.